ВЛИЯНИЕ ПОЛИТИКИ ЗАМЕЩЕНИЯ ЗАПИСЕЙ В КЭШЕ НА ПРОИЗВОДИТЕЛЬНОСТЬ СЕТЕВОГО ПРОЦЕССОРА

Ладыженский Ю.В., Грищенко В.И.
Донецкий национальный технический университет, г. Донецк
кафедра прикладной математики и информатики
E-mail: ly@cs.dgtu.donetsk.ua, grishenko_v@pmi.dgtu.donetsk.ua

Abstract
Ladyzhensky U.V., Grischenko V.I. Influence of the cache replacement policy on network processor productivity. *In this paper three policies are analyzed: LRU, FIFO and Random. Authors use applications from CommBench benchmark to simulate typical network processor workload. Random replacement policy is proposed as most effective on network applications.*

Введение

Постоянное увеличение потоков данных в компьютерных сетях сделало процессоры общего назначения малопригодными для построения маршрутизаторов. Вместо них все чаще используются процессоры пакетной обработки данных (сетевые процессоры, СП).

В СП аппаратно реализованы основные функции обработки данных в информационных сетях (управление очередями, анализ заголовков пакетов, поиск в таблице маршрутизации, вычисление контрольных сумм и др.) [1], одновременно СП, как и процессор общего назначения, предоставляет широкие возможности для программирования.

Одной из основных задач разработки новых сетевых процессоров является определение архитектуры кэш-памяти. В силу ограниченных размеров устройства кэш не может быть сколь угодно большим. В определенный момент дополнительный прирост объема кэш-памяти не дает ожидаемого прироста производительности.

Помимо размера кэша, существенную роль играет и политика замещения устаревших записей. В данной статье анализируется влияние различных алгоритмов замещения на производительность сетевого процессора.

1 Структура кэша

Кэш представляет собой промежуточную быструю память между процессором и ОЗУ. Наибольшее распространение получили три вида кэш-памяти: кэш прямого отображения, множественно-ассоциативный и полностью ассоциативный кэш. В источнике [7] подробно рассмотрены эти типы промежуточной памяти. В предоставленном исследовании анализируется множественно-ассоцативный кэш, как наиболее распространенный в современных процессорах.

Каждый раз, когда вычислительное ядро обращается к оперативной памяти за командой или данными, запрос обрабатывается кэшем. Модуль управления проверяет наличие запрашиваемой записи в кэше и, если она присутствует, возвращает полученные данные без обращения к основной памяти. Если данные не находятся, то такая ситуация называется «промахом кэша». В случае промаха кэш обращается к ОЗУ за запрашиваемой ячейкой, и сохраняет полученные данные в одном из своих блоков. При этом, если все блоки заняты, модуль управления должен выбрать один из них и заменить хранимые в нем данные новой информацией.

Существуют три основных алгоритма замещения данных в кэше [7]: LRU (замещение блока, к которому дольше всего не обращались), FIFO (замещение блоков в порядке очереди) и Random (замещение блоков в случайном порядке).

Наиболее простым в реализации является метод случайного замещения. При его использовании модуль управления обращается к генератору псевдослучайных чисел каждый раз, когда требуется выбрать замещаемый блок.

Алгоритм FIFO подразумевает замещение блоков кэша в циклическом порядке. Для каждого множества блоков хранится отдельный маркер текущего замещаемого блока. Если запрашивается ячейка, отображаемая на это множество, и ее копии нет ни в одном блоке этого множества, то полученные из ОЗУ данные записываются в блок, на который указывает маркер, и значение маркера увеличится на 1.

Наиболее сложным из рассматриваемых является LRU — алгоритм замены блока, к которому дольше всего не было обращений. В этом методе необходимо для каждого блока хранить временную метку последнего обращения и при каждом замещении искать минимальную из меток.

2 Методика анализа производительности СП

Основу применяемого метода исследования [2, 3] составляет эмуляция выполнения приложения на эталонном процессорном ядре и последующий расчет общей производительности сетевого процессора. Обобщенная архитектура СП приведена на рисунке 1. Устройство состоит из однотипных вычислительных ядер, сгруппированных по кластерам. Ядра одного кластера разделяют общий интерфейс к внешней памяти. Каждый процессорный элемент аппаратно реализует многопоточность и имеет собственные кэши команд и данных. Предполагается, что переключение между контекстами потоков осуществляется с нулевой задержкой, т.е. как только один поток переходит в режим ожидания ответа от канала памяти, следующий тут же начинает работу.

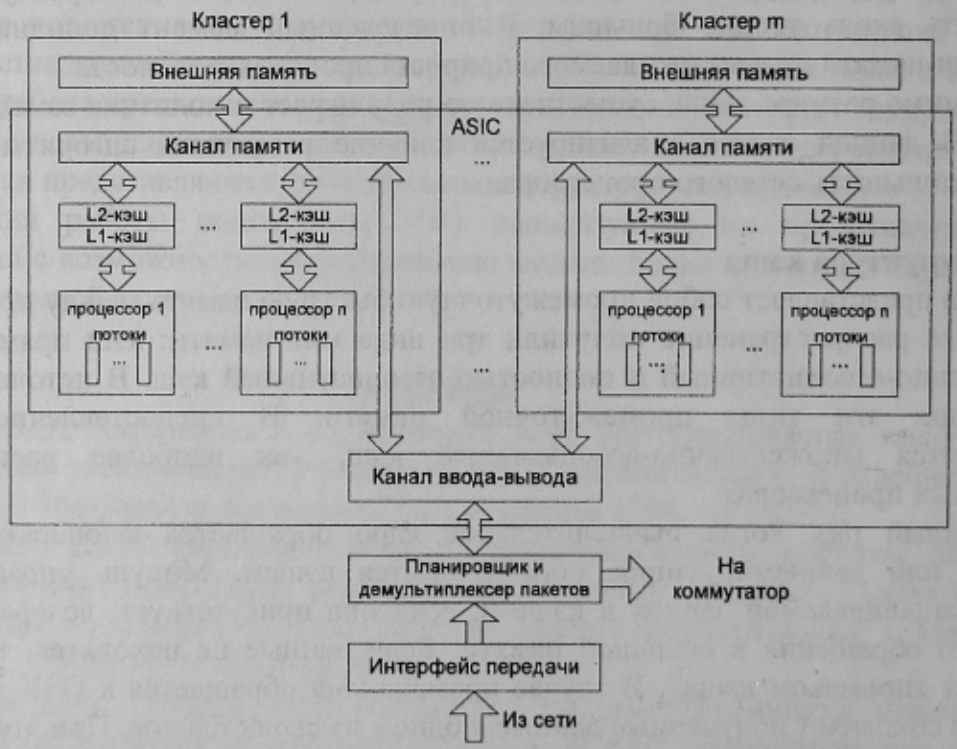

Рисунок 1 – Обобщенная архитектура сетевого процессора

Производительность сетевого процессора оценивается суммой производительностей его вычислительных ядер:

$$IPS = \sum_{j=1}^{m} \sum_{k=1}^{n} \rho_{p_{j,k}} \cdot f_{p_{j,k}} \qquad (1)$$

где $\rho_{p_{j,k}}$ - производительность ядра (j, k), команд/такт, $f_{p_{j,k}}$ - тактовая частота процессора, такт/секунда, m – число кластеров в СП, n – количество процессорных ядер в каждом кластере. Для удобства представления результатов, в статье производительность указывается в миллионах операций в секунду (MIPS).

Производительность отдельного многопоточного процессорного ядра с кэшами данных и команд определяется по формуле [6]:

$$\rho_p(u) = 1 - \frac{1}{\sum_{i=0}^{u} \left(\frac{1}{p_{miss}\tau_{mem}} \right)^i \frac{u!}{(u-i)!}} \quad (2)$$

где ρ_p - загруженность вычислительного ядра, u - число аппаратно поддерживаемых потоков, p_{miss} - интенсивность промахов кэша, τ_{mem} - время доступа к памяти.

Время доступа к памяти учитывает время ожидания запроса в очереди (τ_Q), время физического отклика памяти (τ_{DRAM}) и время передачи данных через кэш ($\tau_{transmit}$):

$$\tau_{mem} = \tau_Q + \tau_{DRAM} + \tau_{transmit} \quad (3)$$

Интенсивность промахов кэшей зависит от приложения, выполняемого на СП:

$$p_{miss} = mi_c + (f_{load} + f_{store}) \cdot md_c \quad (4)$$

где mi_c и md_c - интенсивности промахов кэшей команд и данных соответственно, f_{load} и f_{store} - частота появления в коде программы команд чтения и записи в память.

Имитационное моделирование работы вычислительного ядра осуществляется с использованием системы SimpleScalar [4], которая моделирует выполнение программ на RISC-процессоре. SimpleScalar позволяет определить частоту обращений к ОЗУ, интенсивность промахов кэша и другие характеристики приложения.

В качестве тестового набора приложений в исследовании использовался пакет CommBench [5]. Он охватывает большинство задач, выполняемых на современных маршрутизаторах. Логически эти задачи можно разделить на две группы: приложения, обрабатывающие только заголовки пакетов, и приложения, анализирующие данные, передаваемые по сети. К первой группе относятся задачи фрагментации, маршрутизации и планирования очередей. Во вторую группу входят программы работы с медиа-данными (JPEG), сжатием потока (ZIP), выполнение избыточного кодирования (REED) и шифрование информации (CAST).

3 Задачи обработки заголовков пакетов

Приложения первой группы наиболее распространены на современных маршрутизаторах. Подавляющую часть своего времени работы маршрутизатор решает задачи планирования очередей и поиск следующего узла [8], на который будет отправлен пакет данных. Задача фрагментации [9] относится к поддержке межсетевого взаимодействия и выполняется на граничных маршрутизаторах.

Для задачи планирования очередей на рисунке 2 представлен график зависимости производительности сетевого процессора от размеров кэшей и алгоритма замещения блоков. Для всех трех политик имеет место быстрый рост при увеличении объема кэша данных от 1 до 2 КБ и дальнейшее очень медленное возрастание. Следовательно для данной задачи алгоритм замещения не влияет существенно на производительность.

Для задачи фрагментации использование случайного замещения дает прирост производительности процессора в пределах 8% для кэша команд объемом 1 КБ (см. рис. 3а).

Если увеличить кэш инструкций до 4 КБ (см. рис. 3б), то разница между политиками не превышает 2%. При дальнейшем увеличении кэша команд алгоритмы замещения перестают влиять на производительность СП. Это объясняется тем, что при больших объемах промежуточной памяти существенно уменьшается интенсивность операций замещения данных и использование какого-либо конкретного алгоритма не дает преимуществ.

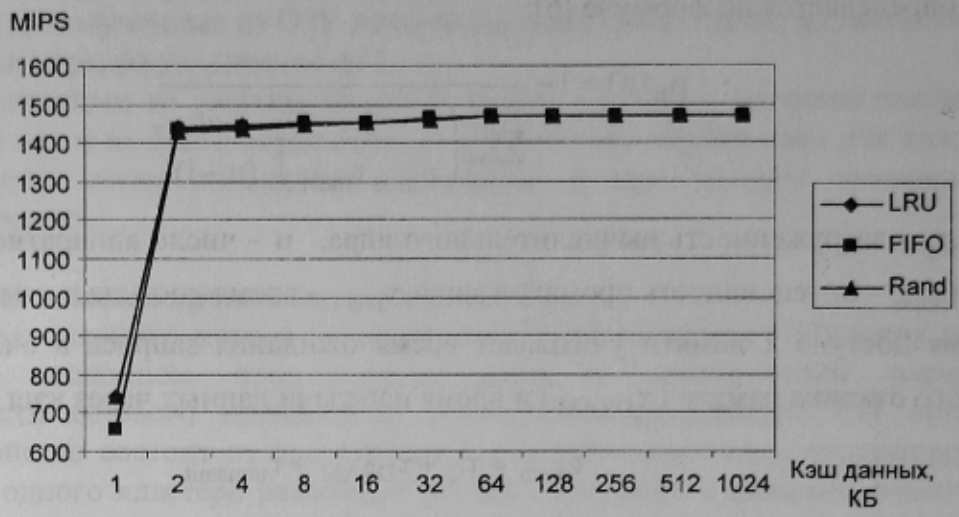

Рисунок 2 – Зависимость производительности СП от размеров кэша данных и алгоритма замещения (кэш команд равен 4 КБ) для задачи планирования очередей (DRR)

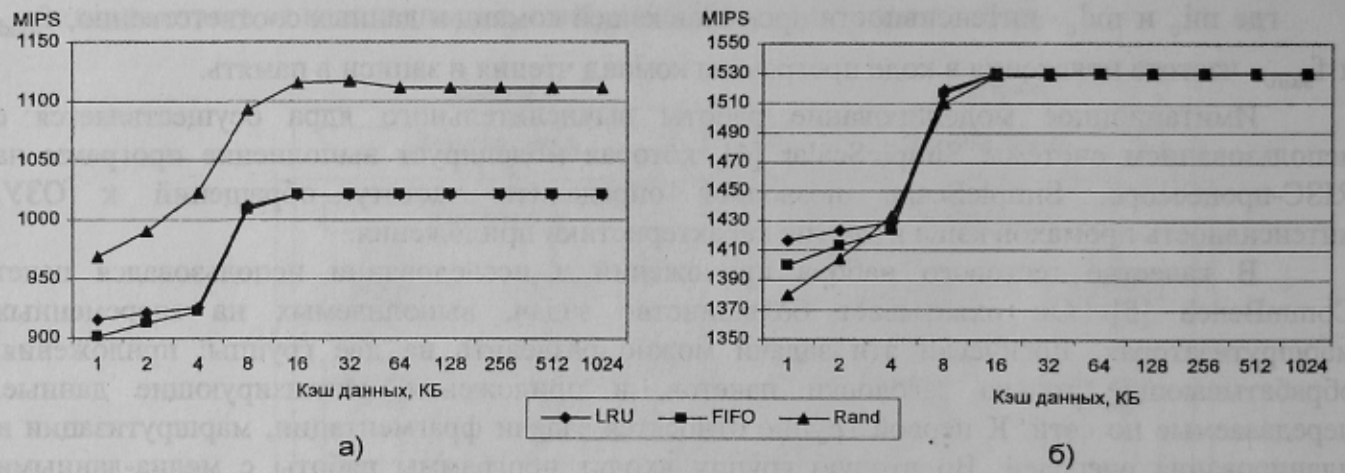

Рисунок 3 – Зависимость производительности СП от размеров кэша данных и алгоритма замещения для задачи фрагментации пакетов (FRAG). а) кэш команд равен 1 КБ; б) кэш команд равен 4 КБ

Анализ задачи маршрутизации подтверждает высказанные предположения. В этом случае использование алгоритма случайной выборки дает прирост производительности порядка 5-10% в зависимости от объема кэша команд. Максимум прироста приходится на кэш инструкций объемом 16 КБ (см. рис. 4), дальнейшее увеличение объема кэша уменьшает различие между алгоритмами.

Проведенный анализ показывает, что изменение политики замещения оказывает существенное влияние на производительность СП только на небольших объемах кэшей. При использовании кэша команд более 16 КБ ни один из алгоритмов замещения не дает заметного прироста производительности системы.

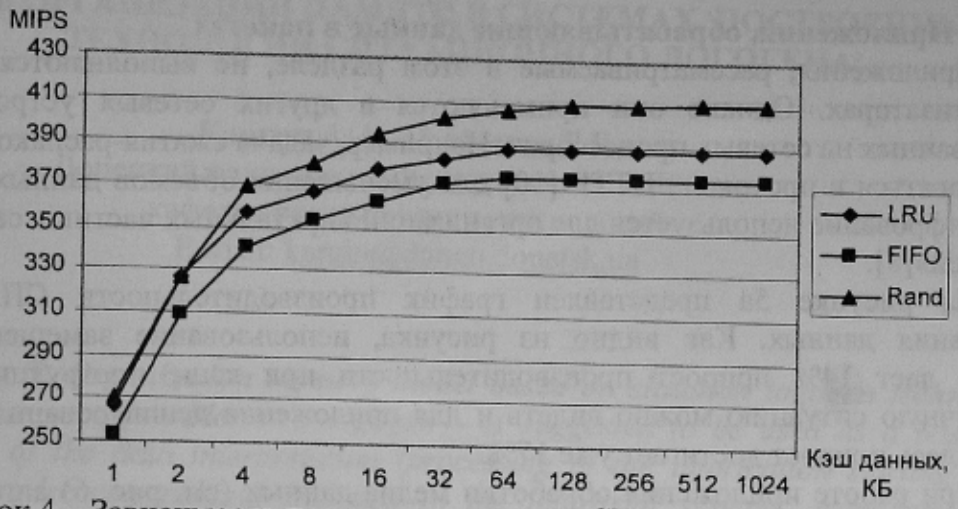

Рисунок 4 – Зависимость производительности СП от размеров кэша данных и алгоритма замещения (кэш команд равен 16 КБ) для задачи маршрутизации (RTR)

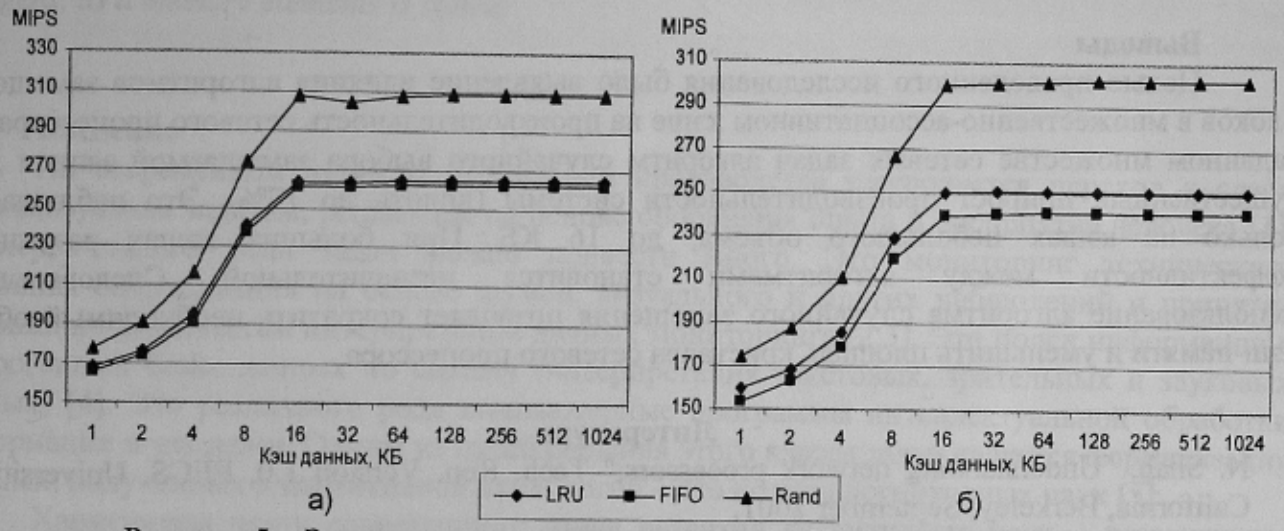

Рисунок 5 – Зависимость производительности СП от размеров кэша данных и алгоритма замещения (кэш команд равен 16 КБ) для задачи шифрования (а) и дешифрования (б) данных по алгоритму CAST

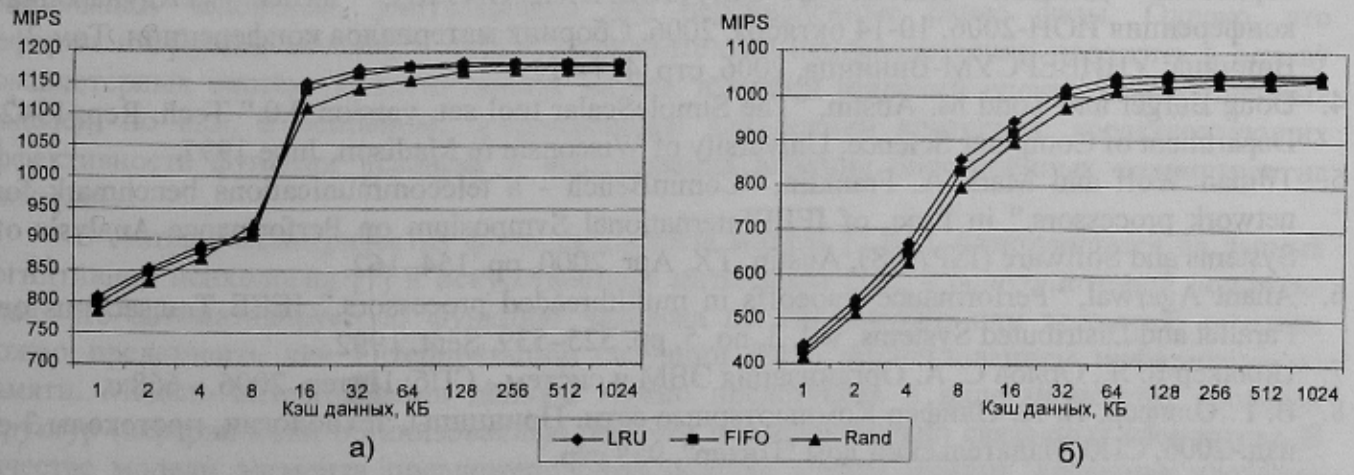

Рисунок 6 – Зависимость производительности СП от размеров кэша данных и алгоритма замещения (кэш команд равен 16 КБ) для задачи сжатия (а) и распаковки (б) медиа-данных с использованием алгоритма JPEG

4 Приложения, обрабатывающие данные в пакетах

Приложения, рассматриваемые в этом разделе, не выполняются на традиционных маршрутизаторах. Однако они применяются в других сетевых устройствах, аппаратно реализованных на сетевых процессорах. Например, задача сжатия-распаковки трафика может использоваться в протоколе HTTP [10] для уменьшения объемов данных, передаваемых по сети. Шифрование используется для организации виртуальных частных сетей и защищенных «туннелей»[8].

На рисунке 5а представлен график производительности СП для приложения шифрования данных. Как видно из рисунка, использование замещения по случайной выборке дает 14% прирост производительности при кэше инструкций объемом 16КБ. Аналогичную ситуацию можно видеть и для приложения дешифрования данных (рис. 5б), однако здесь прирост достигает уже 17%.

При работе приложения обработки медиа-данных (см. рис. 6) алгоритмы замещения не оказывают существенного влияния. Аналогичная ситуация наблюдается и на задачах избыточного кодирования (REED) и сжатия/распаковки информации (ZIP).

Выводы

Целью проведенного исследования было выявление влияния алгоритмов замещения блоков в множественно-ассоциативном кэше на производительность сетевого процессора. На заданном множестве сетевых задач алгоритм случайного выбора замещаемой записи дает существенный прирост производительности системы (вплоть до 17%). Это наблюдается только на кэшах небольшого объема, до 16 КБ. При больших кэшах разница в эффективности между алгоритмами становится незначительной. Следовательно, использование алгоритма случайного замещения позволяет сократить необходимый объем кэш-памяти и уменьшить площадь кристалла сетевого процессора.

Литература

1. N. Shah, "Understanding network processors," Tech. Rep. Version 1.0, EECS, University of California, Berkeley, September 2001.
2. Tilman Wolf, Mark Franklin, “Performance Models for Network Processor Design” IEEE Transactions on Parallel and Distributed Systems, vol. 17, no. 6, pp. 548-561, Jun., 2006.
3. Ладыженский Ю.В., Грищенко В.И. Моделирование сетевых процессоров пакетной обработки данных. ИНТЕРНЕТ-ОБРАЗОВАНИЕ-НАУКА, пятая международная конференция ИОН-2006. 10-14 октября, 2006. Сборник материалов конференции. Том 2. – Винница: УНИВЕРСУМ-Винница, 2006, стр. 417-422.
4. Doug Burger and Todd M. Austin, “The SimpleScalar tool set, version 2.0,” Tech. Rep. 1342, Department of Computer Science, University of Wisconsin in Madison, June 1997.
5. Tilman Wolf and Mark A. Franklin, “CommBench - a telecommunications benchmark for network processors,” in Proc. of IEEEInternational Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, Apr. 2000, pp. 154–162.
6. Anant Agarwal, “Performance tradeoffs in multithreaded processors,” IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 5, pp. 525–539, Sept. 1992.
7. Цилькер Б. Я., Орлов С. А. Организация ЭВМ и систем - СПб: Питер, 2006. - 668 с.
8. В. Г. Олифер, Н. А. Олифер Компьютерные сети. Принципы, технологии, протоколы.3-е изд.-2006, СПб, Издательский дом "Питер", 958 стр.
9. J.Postel. Internet Protocol - DARPA Internet Program Protocol Specification - RFC 791, DARPA, September 1981. http://www.ietf.org/rfc/rfc0791.txt.
10. R. Fielding, J. Gettys et al. Hypertext Transfer Protocol – HTTP/1.1 – RFC 2616, W3C, June 1999. http://www.ietf.org/rfc/rfc2616.txt.